Internationalizing the WAIS Inc Server

Harry Morris (morris@wais.com)
Senior Engineer, WAIS Inc.
12/1/93

There are several steps to be done to internationalize WAIS. By working together, we will learn a great deal about Asian language support and we would like to incorporate the changes into our base code releases so that future maintenance is easier. I will have the pleasure of working on this project with you under the supervision of Brewster Kahle. Please feel free to contact me via email or phone. I would enjoy a trip to work with you in Japan if you find it appropriate. Please feel free to contact me directly or through Brewster Kahle.

We are very interested in understanding the changes necessary to extend WAIS to 2 byte character sets. If we can incorporate the modifications you make into our base code it will be easier to internationalize future versions that we develop.

This document only discusses modifying the server. Clients and the protocol may also need changes. We are beginning development of a major protocol upgrade, so modifications to make it suite your needs are particularly useful.

STEP 1 modify waisparse to tokenize language

WAISParse is the program used during indexing to extract the words from input data. It would be a good place to start, since it is a fairly simple program and it's output is human readable.

The program is given a list of files to process, along with the format of the files. For each file, it reads one line at a time. Based on the file format specified it examines the line and tries to find headlines, fields, and document separators (if there are several documents in the file), and most importantly separates the words in the file. This information is then output in a format described in the file file-parser.txt (which may be in the source code we shipped you, if not, I can email it to you). This format is recognized by waisindex, which uses the output of waisparse to build the index files.

Most of waisparse's work is done in the routine parseFile() in parseFile.c. It reads a line from the file (using the Unix function fgets) and passes it to the appropriate parsing functions for the file format specified by the user. These functions are defined in another file or files (customParsers.c or standardParsers.c and contribParsers.c and localParsers.c depending on the software version you have).

The format name is resolved into the appropriate function pointers by a call to the function findParserNamed(). The function pointers are then called by parseFile(). Finally the job of breaking a line down into words is done by the function writeWords() in parseFile.c.

Most of these routines will have to be modified. In particular, I don't know if fgets() will work on kanjii. WriteWords is the main place to hook in your tokenizer. Finally, the file format specific functions will need looking at since they all assume ASCII.

STEP 2 modify waisindex and waisverify

WAISIndex reads the output of waisparse and builds an inverted index file suitable for searching. I'm not sure what may need to change in waisindex (see step 4 however). The index file formats are defined in the document file-design.txt. Since it is a binary format it is harder to determine if it is correct. Typically it can be tested by searching, but that will not be possible until step 3 is complete. It might be worthwhile to use waisverify to test the index files. WAISverify is not included with the binaries shipped to customers, but the source code is included in the package you have. It is a diagnostic program which reads the index files and dumps them in a human readable form. It also does simple sanity checks on the index files as it runs.

STEP 3 modify search to tokenize language

When the users types a search string into their client, it is packaged up using the Z3950 protocol and sent to the server. The server unpacks the message and parses it into a tree of search terms and Boolean operators. This parsing is done in the file parseDialog.c. The main word separating routine is getNextToken(). Several other routines will also need to be modified to reflect the local names of the Boolean operators (AND/OR/NOT/ADJ).

The waislookup program is a very simple WAIS client which is useful for testing the search routines and the index files. It reads a search string from the Unix command line and passes it straight to the query parsing and search routines.

STEP 4 string handling

Throughout the code, strings are stored in the default C manner, which uses an ASCII 0 to define end of string. Standard C routines are used to manipulate the strings (count length etc.). Also in some cases the program assumes the strings are standard ASCII (1 byte/char) and manipulates them directly. These will all need careful examination.

STEP 5 information retrieval

Once the system is capable of basic indexing and searching, it's IR properties should be addressed. In particular the proximity algorithms need to be checked. They are the routines ANDBooleanProximity() and PostingListScore() in invSearch.c

----

STEP 6 localize error messages and documentation

The error and help messages, as well as the documentation are currently all in English. The should probably be rewritten. Any advice in how best to internationalize the programs' messages would be very helpful, as we have little experience in this area.